Cours 7 : Fichier, exception, module

LIM&BIO

Fichiers

- Pour écrire dans un fichier (crée / efface le fichier !) :

```
fichier = open("nom/du/fichier", "w")
fichier.write("Ce texte va être écrit dans le fichier")
fichier.write("Et ce texte aussi !")
```

- Pour lire le contenu d'un fichier :

```
fichier = open("nom/du/fichier", "r")
print "le fichier contient :", fichier.read()
```

- On peut aussi découper le contenu du fichier en ligne :

```
fichier = open("nom/du/fichier", "r")
contenu = fichier.read()
lignes = contenu.split("\n")
print lignes -> ["ligne n°1", "ligne n°2", "ligne n°3"]
```

- Les échanges réseaux fonctionnent de la même manière

Fichiers

- Exercice :
  - Lire une séquence d'ARNm depuis le fichier ./arn, la traduire et enregistrer la séquence protéique dans le fichier ./proteine
    - On utilisera la fonction de traduction écrite précédemment

Fichiers

- Exercice :
  - Lire une séquence d'ARNm depuis le fichier ./arn, la traduire et enregistrer la séquence protéique dans le fichier ./proteine
    - On utilisera la fonction de traduction écrite précédemment

```
fichier = open("./arn")
arn = fichier.read()
proteine = traduction(arn)
fichier = open("./proteine", "w")
fichier.write(proteine)
```

Fichiers

- Exercice :
  - Écrire un programme pour créer le dictionnaire du code génétique à partir du fichier code_genetique.txt, qui contient le code génétique sous la forme suivante :

UUU F
UUC F
UUA L
UUG L
UCU S
[codon acideaminé...]

Fichiers

- Exercice :
  - Écrire un programme pour créer le dictionnaire du code génétique à partir du fichier code_genetique.txt, qui contient le code génétique sous la forme suivante :

UUU F
UUC F
[codon acideaminé...]

```
code_genetique = {}
fichier = open("./code_genetique.txt")
contenu = fichier.read()
lignes = contenu.split("\n")
for ligne in lignes:
        mots = ligne.split(" "):
        code_genetique[mots[0]] = mots[1]
```

Exceptions

- Les exceptions correspondent aux erreurs :

```
print 1 + "2"
Traceback (most recent call last):
  File "<pyshell#92>", line 1, in -toplevel-
    print 1 + "2"
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- Il est possible de «récupérer» les erreurs :

```
try:
        code susceptible de déclencher une erreur
except nom_de_l'erreur :
        code à effectuer en cas d'erreur

except:          récupère toutes les erreurs
        pass     ne fait rien
```

Exceptions

- Principales exceptions :
  - TypeError : mauvais type de donnée
  - ValueError : mauvaise valeur
  - ZeroDivisionError : division par 0
  - SyntaxError : erreur de syntaxe
  - KeyboardInterrupt : arrêt manuel du programme (contrôle-C)
- Exemple :

```
try:
        imc = poids / (taille * taille)
except ZeroDivisionError:
        imc = 0.0
```

- Il est possible de déclencher soi-même des erreurs :

```
raise ValueError("message d'erreur")
```

Exceptions

- Exercice :
  - Écrire une fonction qui retourne 1 si deux séquences d'ADN sont complémentaires, 0 sinon, et qui déclenche une exception si les deux séquences n'ont pas la même longueur

Exceptions

- Exercice :
  - Écrire une fonction qui retourne 1 si deux séquences d'ADN sont complémentaires, 0 sinon, et qui déclenche une exception si les deux séquences n'ont pas la même longueur

```
def complementaire(adn1, adn2):
    if len(adn1) != len(adn2):
        raise ValueError("les deux séquences n'ont pas la
                                                même longueur")
    for i in range(len(adn1)):
        if (adn1[i] == "a") and (adn2[i] != "t" ): return 0
        if (adn1[i] == "t" ) and (adn2[i] != "a"): return 0
        if (adn1[i] == "c") and (adn2[i] != "g"): return 0
        if (adn1[i] == "g") and (adn2[i] != "c"): return 0
    return 1
```

Exceptions

- Exercice :
  - Écrire une fonction qui demande à l'utilisateur d'entrer un numéro de chromosome et le retourne, et convertit ce numéro en entier si cela est possible

Exceptions

- Exercice :
  - Écrire une fonction qui demande à l'utilisateur d'entrer un numéro de chromosome et le retourne, et convertit ce numéro en entier si cela est possible

```
def demander_chromosome():
    chromosome = raw_input("Entrez un numéro de chromosome : ")
    try:
            chromosome = int(chromosome)
    except:
            pass
    return chromosome
```

Modules

- Des modules (ou «package») Python permettent d'accéder à de nouvelles fonctions
  - Python contient de nombreux modules
  - D'autres modules peuvent être télécharger sur Internet
    - Le site http://pypi.python.org recense un grand nombre de modules
    - BioPython pour la bio-informatique
  - Il est possible d'écrire ses propres modules
    - Avantage : permet de partager des fonctions entre plusieurs programmes

Modules

- Utilisation d'un module :

  import *nom_du_module*

  nom_du_module.*nom_de_la_fonction*()

- Utilisation d'un module (autre possibilité) :

  from *nom_du_module* import *

  nom_de_la_fonction()

- Exemple :

  import os

  print os.listdir("/home/jblamy")

  -> ["tmp", "téléchargements", "cours",...]

Modules

- Les modules peuvent être imbriqués les uns dans les autres :
- Exemple : le module path est imbriqué dans le module os

```
import os.path
os.path.dirname("/home/jblamy/cours/cours2_python.sxi")
-> "/home/jblamy/cours"
```

Modules

- Quelques modules inclus dans Python :

os	fonctions liés au système d'exploitation (fichiers...)
os.path	fonctions pour manipuler des noms de fichiers
sys	système, fonctions propres à Python
math	fonctions mathématiques avancées (sin, cos,...)
random	génération de nombres aléatoires (= au hasard)
string	fonctions avancées sur les chaînes de caractères
re	recherche de motifs complexes dans des chaînes
webbrowser	gestion du navigateur internet
zipfile	création / décompression de fichier .zip
Tkinter	interface graphique

La documentation complète de Python et de tous les modules inclut dans Python est téléchargeable ici :

http://www.python.org/doc/2.4.4/modindex.html

Modules

- Le module math :
  - math.cos(x)
  - math.sin(x)
  - math.tan(x)
  - math.log(x) : ln mathématique
  - math.log10(x) : log mathématique
  - math.exp(x) : e^x (exponentielle)
  - math.pow(x, y) : x^y (x ** y)
  - math.sqrt(x) : racine de x

- Le module random :
  - random.random() : un nombre au hasard entre 0.0 et 1.0
  - random.randint(x, y) : un entier au hasard entre x et y
  - random.choice(liste) : un élément de la liste au hasard

Modules

- Exercice :
  - Écrire une fonction qui retourne une chaîne d'ADN aléatoire, avec un nombre de base donné.

Modules

- Exercice :
  - Écrire une fonction qui retourne une chaîne d'ADN aléatoire, avec un nombre de base donné.

```
import random

def adn_aleatoire(nb_base):
        bases = ["a", "t", "c", "g"]
        adn = ""
        for i in range(nb_base):
                adn = adn + random.choice(bases)
        return adn
```

Modules

- Pour créer ses propres modules :
  - Créer un fichier mon_module.py
  - Mettre dans ce fichier les fonctions du module
  - Importer ce module avec import mon_module dans un script placé dans le même répertoire
- Exercice :
  - Créer un module imc avec la fonction indice_de_masse_corporelle qui calcule l'indice de masse corporelle
  - Créer un script test.py qui utilise ce module pour afficher mon indice de masse corporelle (poids : 64 Kg, taille 1m70)

Modules

- Exercice :
  - Créer un module imc avec la fonction indice_de_masse_corporelle qui calcule l'indice de masse corporelle

dans le fichier imc.py :

```
def indice_de_masse_corporelle(poids, taille):
        return poids / (taille * taille)
```

  - Créer un script test.py qui utilise ce module pour afficher mon indice de masse corporelle (poids : 64 Kg, taille 1m70)

dans le fichier test.py :

```
import imc
print imc.indice_de_masse_corporelle(64.0, 1.70)
```

Modules

- Pour créer ses propres modules imbriqués :
  - Les modules imbriqués correspondent à des répertoires
  - Créer un répertoire mon_module
  - Créer un fichier __init__.py
  - Mettre dans ce fichier les fonctions du module
  - Créer un fichier mon_sous_module.py
  - Mettre dans ce fichier les fonctions du sous_module
  - Importer ce module avec :
    import mon_module.mon_sous_module

Modules

- Exemple : modules clinique et clinique.imc :
  - Créer un répertoire clinique
  - Dans clinique/__init__.py :

  [rien, mais le fichier doit exister !]
  - Dans clinique/imc.py :

```
def indice_de_masse_corporelle(poids, taille):
        return poids / (taille * taille)
```

  - Dans script.py :

```
import clinique.imc
print clinique.imc.indice_de_masse_corporelle(64.0, 1.70)
```

Examen

- Exercice du même type que ceux fait en TD / TP
- Lors de l'examen, sont autorisés :
  - Les notes prises en cours / TD / TP
  - Les impressions des support de cours / TD
  - Les sujets de TP
- sont interdits :
  - Livres, calculatrice, ordinateur, téléphone,...